Variable and value elimination in binary constraint satisfaction via forbidden patterns ☆

David A. Cohen [a], Martin C. Cooper [b,*], Guillaume Escamocher [c], Stanislav Živný [d]

[a] Dept. of Computer Science, Royal Holloway, University of London, UK
[b] IRIT, University of Toulouse III, 31062 Toulouse, France
[c] Insight Centre for Data Analytics, University College Cork, Ireland
[d] Department of Computer Science, University of Oxford, UK

A B S T R A C T

Variable or value elimination in a constraint satisfaction problem (CSP) can be used in preprocessing or during search to reduce search space size. A variable elimination rule (value elimination rule) allows the polynomial-time identification of certain variables (domain elements) whose elimination, without the introduction of extra compensatory constraints, does not affect the satisfiability of an instance. We show that there are essentially just four variable elimination rules and three value elimination rules defined by forbidding generic sub-instances, known as irreducible existential patterns, in arc-consistent CSP instances. One of the variable elimination rules is the already-known Broken Triangle Property, whereas the other three are novel. The three value elimination rules can all be seen as strict generalisations of neighbourhood substitution.

1. Introduction

Constraint satisfaction has proved to be a useful modelling tool in a variety of contexts, such as scheduling, timetabling, planning, bio-informatics and computer vision [17,24,29]. Dedicated solvers for constraint satisfaction are at the heart of the programming paradigm known as constraint programming. Theoretical advances on CSPs can thus potentially lead to the improvement of generic combinatorial problem solvers.

In the CSP model we have a number of variables, each of which can take values from its particular finite domain. Certain sets of the variables are constrained in that their simultaneous assignments of values is limited. The generic problem in which these sets of variables, known as the constraint scopes, are all of cardinality at most two, is known as binary constraint satisfaction. We are required to assign values to all variables so that every constraint is satisfied. Complete solution algorithms for constraint satisfaction are not polynomial time unless $P = \text{NP}$, since the graph colouring problem, which is NP-complete, can be reduced to binary constraint satisfaction [17]. Hence we need to find ways to reduce the search space.

☆ A preliminary version of part of this work appeared in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.
* Corresponding author.
E-mail addresses: dave@cs.rhul.ac.uk (D.A. Cohen), cooper@irit.fr (M.C. Cooper), guillaume.escamocher@insight-centre.org (G. Escamocher),

Search algorithms for constraint problems usually proceed by transforming the instance into a set of subproblems, for example, by selecting a variable and assigning to it successively each value from its domain. This naive backtracking approach is recursive and explores the search tree of partial assignments in a depth first manner. Even though the backtracking algorithm can take exponential time it is often effective in practice thanks to intelligent pruning techniques.

There are many ways to improve naive backtracking by pruning the search space in ways that cannot remove solutions. This is done by avoiding searching exhaustively in all generated subproblems when certain kinds of discovered obstruction to solution exists. Such techniques include Back-marking, Back-jumping, Conflict-Directed Back-jumping [6,28]. As well as these look-back techniques it is also possible to look ahead by propagating the consequences of early decisions or of the discovered structure. Of these look-ahead techniques the most common is to maintain the local consistency property called generalised arc-consistency (GAC) [5]. This technique identifies certain values for variables that cannot possibly form part of a solution.

Of course, savings can also be made if we are able to eliminate variables from a sub-problem. Since backtracking is of exponential time complexity, the elimination of variables and values (domain elements) to reduce instance size can in the best case reduce search time by an exponential factor. To maintain the soundness of search we require that such eliminations do not change the satisfiability of the instance. Invariance of satisfiability, which we study in the present paper, is a weaker property than the invariance of the set of solutions guaranteed by consistency techniques such as GAC. However, detection of non-satisfiability is the essential role of look-ahead techniques, since this allows pruning during search. Thus satisfiability-preserving reduction techniques (which do not necessarily preserve solutions) may prove useful even when the aim is to discover one or all solutions. In fact, we show that all the techniques presented in this paper, although they do not preserve solutions, allow a solution to the original instance to be reconstructed very efficiently.

1.1. Simplification by variable and value elimination

We consider an instance I of the CSP viewed as a decision problem. Suppose that x is a variable of I and that, whenever there is some valid assignment to all variables except x, there is a solution to the whole instance; in this case, we can safely remove variable x from I. One of the questions we address in this paper is how to identify such variables?

Variable elimination has been considered before in the literature. It is well known that in an arc-consistent binary CSP instance, a variable x which is constrained by only one other variable y can be eliminated; by the definition of arc consistency, each assignment to y is compatible with some assignment to x. It has been observed that a more general property, called the (local) Broken Triangle Property (lBTP) [11], if it holds at some variable, allows us to eliminate that variable. One way of stating the lBTP is that there is no pair of compatible assignments to two other variables y, z which have opposite compatibilities with two assignments to x. The closure of a binary CSP instance under the elimination of all variables that satisfy the lBTP is unique and can be found in $O(ncd^3)$ time, where n is the number of variables, c the number of constraints and d the maximum domain size, which may well prove effective when compared to the exponential cost of backtracking. The more general local min-of-max extendable property (lMME) allows us to eliminate more variables than the lBTP, but requires the identification of a particular domain order. Unfortunately, this domain order is NP-hard to discover [11] for unbounded domain size, and so the lMME is less likely to be effective in practice.

An alternative to simple variable elimination is used in Bucket Elimination [23]. In this algorithm a variable v is not simply eliminated. Instead it is replaced by a constraint on its neighbourhood (the set of variables constrained by v). This new constraint precisely captures those combinations of assignments to the neighbourhood of v which can be extended to a consistent assignment to v. Such an approach may generate high-order constraints, which are exponentially hard to process and to store. The arity can be bounded by the induced treewidth of the instance, but this still limits the applicability of Bucket Elimination. In the present paper we restrict our attention to the identification of variable elimination strategies which do not require the addition of compensatory constraints.

The elimination of domain elements is an essential component of constraint solvers via generalised arc consistency (GAC) operations. GAC eliminates domain elements that cannot be part of any solution, thus conserving all solutions. An alternative approach is the family of elimination rules based on substitution: if all solutions in which variable v is assigned value b remain solutions when the value of variable v is changed to another value a, then the value b can be eliminated from the domain of variable v while conserving at least one solution (if the instance is satisfiable). The most well-known polynomial-time detectable substitution operation is neighbourhood substitution [18]. The value elimination rules described in this paper go beyond the paradigms of consistency and substitution; we only require that the instance obtained after elimination of a value from a domain has the same satisfiability as the original instance.

We study rules for simplifying binary CSP instances based on properties of the instance at the microstructure level. The term microstructure was first given a formal definition by Jégou [22]: if I is a binary CSP instance, then its *microstructure* is a graph $\langle A, E\rangle$ where A is the set of possible variable-value assignments and E is the set of pairs of compatible variable-value assignments. Solutions to I are in one-to-one correspondence with the n-cliques of the microstructure of I and with the size-n independent sets of the microstructure complement of I. The *chromatic number* of a graph is the smallest number of colours required to colour its vertices so that no two adjacent vertices have the same colour. A graph G is *perfect* if for every induced subgraph H of G, the chromatic number of H is equal to the size of the largest clique contained in H. Since a maximum clique in a perfect graph can be found in polynomial time [21], the class of binary CSP instances with a perfect microstructure is tractable [30]. Perfect graphs can also be recognised in polynomial time [16]. An instance

of the minimum-cost homomorphism problem (MinHom) is a CSP instance in which weights are associated with each variable-value assignment and the aim is to find a solution which minimises the sum of the weights. Takhanov [31] gave a dichotomy for tractable conservative constraint languages for MinHom which uses the fact that an instance of binary MinHom can be solved in polynomial time if its microstructure is perfect. El Mouelhi et al. [26] make the observation that if the microstructure has a bounded number of maximal cliques then the instance will be solved in polynomial time by classical algorithms such as Forward Checking or Really Full Lookahead and hence by CSP solvers.

Simple rules for variable or value elimination based on properties of the microstructure are used by Beigel and Eppstein [4] in their algorithms with low worst-case time bounds for such NP-complete problems as 3-COLOURING and 3SAT. Such simplification operations are an essential first step before the use of decompositions into subproblems with smaller domains. A similar approach allows Angelsmark and Thapper [3] to reduce the problem of finding a minimum weighted independent set in the microstructure complement to the problem of counting the number of solutions to a 2SAT instance. Thus, the variable and value elimination rules we present in this paper may find not only practical applications in solvers but also theoretical applications.

1.2. Our contribution

In this paper we characterise those local conditions under which we can eliminate variables or values in binary CSPs while preserving satisfiability of the instance, without the need to add compensating constraints. By local conditions we mean configurations of variables, values and constraints which *do not* occur. That is, we will identify (local) obstructions to variable or value elimination. We will call such constructions variable elimination or value elimination patterns.

Surprisingly we find that there are precisely four (three) essentially different local patterns whose absence permits variable (value) elimination. Searching for these local patterns takes polynomial time and need only be done during the pre-processing stage, before search. Any discovered obstructions to elimination can be effectively monitored during subsequent search using techniques analogous to watched literals [20]. Whenever a variable (value) no longer participates in any obstruction patterns it can safely be eliminated.

We show that after a sequence of variable and value eliminations it is always possible to reconstruct a solution to the original instance from a solution to the reduced instance in low-order polynomial time.

2. Definitions

When certain kinds of local obstructions are not present in a binary CSP instance, variable or value elimination is possible. Such obstructions are called quantified patterns. A pattern can be seen as a generalisation of the concept of a constraint satisfaction instance that leaves the consistency of some assignments to pairs of variables undefined.

Definition 2.1. A *pattern* is a four-tuple $\langle X, D, A, \mathrm{cpt}\rangle$ where:

- X is a finite set of *variables*;
- D is a finite set of *values*;
- $A \subseteq X \times D$ is the set of possible *assignments*; the *domain* of $v \in X$ is its non-empty set $\mathcal{D}(v)$ of possible values: $\mathcal{D}(v) = \{a \in D \mid \langle v, a\rangle \in A\}$; and
- cpt is a partial *compatibility function* from the set of unordered pairs of assignments $\{\{\langle v, a\rangle, \langle w, b\rangle\} \mid v \neq w\}$ to $\{\mathtt{TRUE}, \mathtt{FALSE}\}$; if $\mathrm{cpt}(\langle v, a\rangle, \langle w, b\rangle) = \mathtt{TRUE}$ (resp., $\mathtt{FALSE}$) we say that $\langle v, a\rangle$ and $\langle w, b\rangle$ are *compatible* (resp., *incompatible*).

A *quantified pattern* is a pattern P with a distinguished variable, $\overline{v}(P)$ and a subset of existential values $e(P) \subseteq \mathcal{D}(\overline{v}(P))$.

A *flat quantified pattern* is a quantified pattern for which $e(P)$ is empty. An *existential pattern* is a quantified pattern P for which $e(P)$ is non-empty. An existential pattern P may also have a distinguished value $\overline{val}(P) \in e(P)$.

When the context variable v is clear we use the value a to denote the assignment $\langle v, a\rangle$ to v. We will often simplify notation by writing $\mathrm{cpt}(p, q)$ for $\mathrm{cpt}(\{p, q\})$. We will also use the terminology of graph theory, since a pattern can be viewed as a labelled graph: if $\mathrm{cpt}(p, q) = \mathtt{TRUE}$ (resp., $\mathtt{FALSE}$), then we say that there is a *compatibility* (resp., *incompatibility*) *edge* between p and q.

We will use a simple figurative drawing for patterns. Each variable will be drawn as an oval containing dots for each of its possible assignments. Pairs in the domain of the function cpt will be represented by lines between values: solid lines for compatibility and dashed lines for incompatibility. The distinguished variable ($\overline{v}(P)$) and any existential values in $e(P)$ will be indicated by an $\exists$ symbol. Examples of patterns are shown in Fig. 1 and Fig. 2.

We are never interested in the names of variables nor the names of the domain values in patterns. So we define the following equivalence.

Definition 2.2. Two patterns P and Q are *equivalent* if they are isomorphic, i.e. if they are identical except for possible injective renamings of variables and assignments which preserve $\mathcal{D}$, cpt, $\overline{v}$, e and $\overline{val}$.

A pattern can be viewed as a CSP instance in which not all compatibilities are defined. We can thus refine patterns to give a definition of a (binary) CSP instance.

Definition 2.3. A *binary CSP instance* P is a pattern $\langle X, D, A, \text{cpt}\rangle$ where cpt is a total function, i.e. the domain of cpt is precisely $\{\{\langle v, a\rangle, \langle w, b\rangle\} \mid v \neq w, a \in \mathcal{D}(v), b \in \mathcal{D}(w)\}$.

- The *relation* $R_{v,w} \subseteq \mathcal{D}(v) \times \mathcal{D}(w)$ on $\langle v, w\rangle$ is $\{\langle a, b\rangle \mid \text{cpt}(\langle v, a\rangle, \langle w, b\rangle) = \texttt{TRUE}\}$.
- A *partial solution* to P on $Y \subseteq X$ is a mapping $s : Y \to D$ where, for all $v \neq w \in Y$ we have $\langle s(v), s(w)\rangle \in R_{v,w}$.
- A *solution* to P is a partial solution on X.

For notational simplicity we have assumed that there is exactly one binary constraint between each pair of variables. In particular, this means that the absence of a constraint between variables v, w is modelled by a complete relation $R_{v,w} = \mathcal{D}(v) \times \mathcal{D}(w)$ allowing every possible pair of assignments to v and w. We say that there is a *non-trivial* constraint on variables v, w if $R_{v,w} \neq \mathcal{D}(v) \times \mathcal{D}(w)$.

In practice, when solving CSP instances we prune the domains of variables in such a way as to maintain all solutions.

Definition 2.4. Let $P = \langle X, D, A, \text{cpt}\rangle$ be a CSP instance. An assignment $\langle v, a\rangle \in A$ to variable v is called *arc consistent* if, for all variables $w \neq v$ in X there is some assignment $\langle w, b\rangle \in A$ compatible with $\langle v, a\rangle$.

The CSP instance $\langle X, D, A, \text{cpt}\rangle$ is called *arc consistent* if every assignment in A is arc consistent.

Assignments that are not arc-consistent cannot be part of a solution so can safely be removed. There are optimal $O(cd^2)$ algorithms for establishing arc consistency which repeatedly remove such values [5], where c is the number of non-trivial constraints and d the maximum domain size. Hence, for the remainder of this paper we will assume that all CSP instances are arc-consistent.

In this paper we are concerned with variable elimination characterised by forbidden patterns. We now define what this means.

Definition 2.5. We say that a variable x can be *eliminated* in the CSP instance $\langle X, D, A, \text{cpt}\rangle$ if, whenever there is a partial solution on $X \setminus \{x\}$ there is a solution.

In order to use (the absence of) patterns for variable elimination we need to define what we mean when we say that a quantified pattern occurs at variable x of a CSP instance. We define occurrence in terms of reductions on patterns. The definitions of occurrence and reduction between quantified patterns extend definitions previously given for non-quantified patterns [8].

Definition 2.6. Let $P = \langle X, D, A, \text{cpt}\rangle$ be any pattern.

- We say that a pattern $P' = \langle X', D', A', \text{cpt}'\rangle$ is a *sub-pattern* of P if $X' \subseteq X$, $A' \subseteq A$ and $\forall p, q \in A'$, either $\text{cpt}'(p, q) = \text{cpt}(p, q)$ or $\text{cpt}'(p, q)$ is undefined.
  If, furthermore, P' is quantified then we require that P is quantified and that $\bar{v}(P') = \bar{v}(P)$ and $\underline{e}(P') \subseteq e(P)$. If P' has a distinguished value then we require that P also has a distinguished value and that $\underline{val}(P') = \underline{val}(P)$.
- Values $a, b \in \mathcal{D}(v)$ are *mergeable* in a pattern if there is no assignment $p \in A$ for which $\text{cpt}(\langle v, a\rangle, p)$, $\text{cpt}(\langle v, b\rangle, p)$ are both defined and $\text{cpt}(\langle v, a\rangle, p) \neq \text{cpt}(\langle v, b\rangle, p)$. In a quantified pattern, for a to be merged into b, we also require that $a \in e(P)$ only if $b \in e(P)$.
  When $a, b \in \mathcal{D}(v)$ are mergeable we define the merge reduction $\langle X, D, A \setminus \{\langle v, a\rangle\}, \text{cpt}'\rangle$, in which a is merged into b, by the following compatibility function:

$$\text{cpt}'(p, q) = \begin{cases} \text{cpt}(\langle v, a\rangle, q) & \text{if } p = \langle v, b\rangle \text{ and } \text{cpt}(p, q) \text{ undefined,} \\ \text{cpt}(p, q) & \text{otherwise.} \end{cases}$$

- A *dangling assignment* p of P is any assignment for which there is at most one assignment q for which $\text{cpt}(p, q)$ is defined, and furthermore (if defined) $\text{cpt}(p, q) = \texttt{TRUE}$. If P is quantified, then we also require that $p \notin \bar{v}(P) \times e(P)$. For any dangling assignment p, we define the dangling reduction $\langle X, D, A', \text{cpt} \restriction_{A' \times A'}\rangle$ where $A' = A \setminus \{p\}$.
- A *reduction* of a pattern P is a pattern obtained from P by a sequence of merge and dangling reductions. An *irreducible pattern* is one on which no merge or dangling reductions can be performed.

To illustrate the notions introduced in Definition 2.6, consider the patterns in Fig. 1. Pattern P_1 is a sub-pattern of P_2 which is itself a sub-pattern of P_3. In pattern P_2, the values $a, b \in \mathcal{D}(x)$ are mergeable: merging a into b produces the pattern P_4. In the pattern P_3, the values $a, b \in \mathcal{D}(x)$ are not mergeable since $\text{cpt}(\langle x, a\rangle, \langle z, d\rangle)$ and $\text{cpt}(\langle x, b\rangle, \langle z, d\rangle)$ are both defined but are not equal. In pattern P_2, $\langle x, a\rangle$ is a dangling assignment: applying the dangling reduction to this assignment

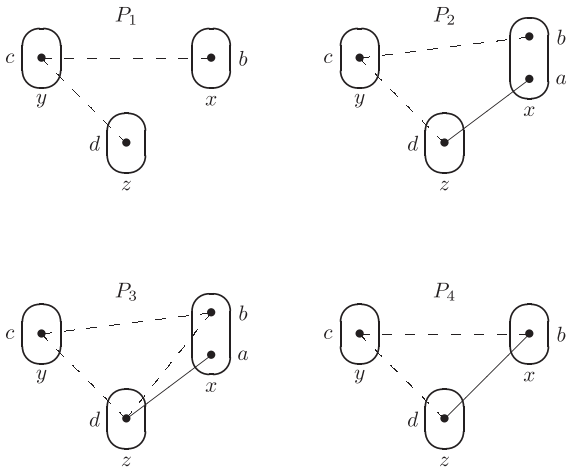

Fig. 1. Examples illustrating the notions of sub-pattern, merging and dangling assignment.

in P_2 produces P_1. Let P_2' be identical to P_2 except that P_2' is a quantified pattern with $\overline{v}(P_2') = \{x\}$ and $e(P_2') = \{a\}$. Then P_2 is a sub-pattern of P_2', but P_2' is not a sub-pattern of P_2. In the domain of x in P_2', b can be merged into a but a cannot be merged into b since $a \in e(P_2')$ but $b \notin e(P_2')$. Furthermore, the assignment $\langle x, a\rangle$ is not a dangling assignment in P_2' since a is an existential value for $\overline{v}(P_2') = x$.

Now we want to define when a quantified pattern occurs at a variable in a CSP instance, in order to characterise those patterns whose *non-occurrence* allows this particular variable to be eliminated. We define the slightly more general notion of occurrence of a pattern in another pattern. Recall that a CSP instance corresponds to the special case of a pattern whose compatibility function is total. Essentially we want to say that pattern P occurs in pattern Q if P is homomorphic to a sub-pattern of Q via an injective renaming of variables and a (possibly non-injective) renaming of assignments [7]. However, we find it simpler to define occurrence using the notions of sub-pattern, reduction and equivalence. We first make the observation that dangling assignments in a pattern provide no useful information since we assume that all CSP instances are arc consistent, which explains why dangling assignments can be eliminated from patterns.

We can then define occurrence in terms of reduced patterns.

Definition 2.7. We say that a pattern P *occurs* in a pattern Q (and that Q *contains* P) if some reduction of P is equivalent to a sub-pattern of Q.

If Q is a CSP instance, then the quantified pattern P *occurs at variable* x *of* Q if some reduction of P is equivalent to a sub-pattern of Q and x is the variable of the sub-pattern of Q corresponding to $\overline{v}(P)$.

We say that the quantified pattern P *occurs at variable* x *of* Q *with value mapping* $m : e(P) \to \mathcal{D}(x)$ if the values of variable x corresponding to each $a \in e(P)$ are given by the mapping m.

A variable elimination pattern is defined in terms of occurrence of a pattern in a CSP instance.

Definition 2.8. A quantified pattern is a *variable elimination pattern (var-elim pattern)* if, whenever the pattern does not occur at a variable x in an arc-consistent CSP instance I for at least one injective value mapping, x can be eliminated in I (in the sense of Definition 2.5).

A non-quantified pattern (i.e. a pattern without a distinguished variable) is a var-elim pattern if, whenever the pattern does not occur in an arc-consistent CSP instance, *any* variable can be eliminated in I.

The notion of non-quantified var-elim patterns is necessary for some of our proofs, but for practical applications we are interested in finding quantified (and, in particular, existential) var-elim patterns. Existential patterns may allow more variables to be eliminated than flat quantified patterns. For example, as we will show later, the patterns snake and ∃snake shown in Fig. 2 are both var-elim patterns, but the latter allows more variables to be eliminated since we only require that it does not occur on a single value in the domain of the variable to be eliminated.

Example 2.1. Suppose that we can assign value 0 to a subset S of the variables of an instance, without restricting the assignments to any other variables. Furthermore suppose that, within S, 0 is only compatible with 0. The var-elim pattern ∃invsubBTP, shown in Fig. 2, allows us to eliminate all variables in S, without having to explicitly search for S. This is because the pattern does not occur for the mapping $a \mapsto 0$. The flat variant (invsubBTP) would not allow these eliminations.

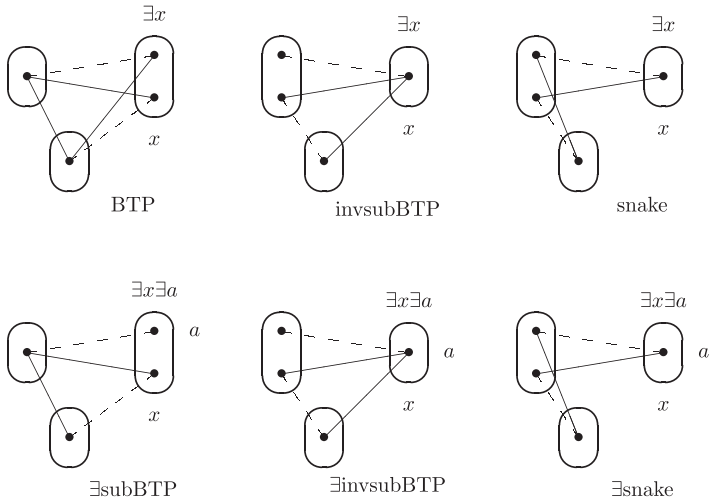

Fig. 2. Variable elimination patterns.

We conclude this section with the simple observation that var-elim patterns define tractable classes. It takes polynomial time to establish arc consistency and to detect (by exhaustive search) the non-occurrence of a var-elim pattern. Hence it takes polynomial time to identify arc-consistent CSP instances for which all variables can be eliminated one by one by a var-elim pattern P. Such instances are solvable in a greedy fashion.

Hence we are able to significantly extend the list of known tractable classes defined by forbidden patterns since among known tractable patterns, namely BTP [11], 2-constraint patterns [8], pivots [7] and JWP [9], *only* BTP (and its sub-patterns) allow variable elimination.

Indeed, a general hybrid tractable class can be defined: the set of binary CSP instances which fall in some known tractable class after we have performed all variable (and value) eliminations defined by the rules given in this paper.

3. Variable elimination by forbidden patterns

In this paper we characterise irreducible var-elim patterns. There are essentially just four (together with their irreducible sub-patterns): the patterns BTP, ∃subBTP, ∃invsubBTP and ∃snake, shown in Fig. 2. We begin by showing that each of these four patterns allows variable elimination. Forbidding BTP is equivalent to the already-known local Broken Triangle Property (lBTP) [11] mentioned in Section 1.1.

Theorem 3.1. *The patterns BTP, ∃subBTP, ∃invsubBTP and ∃snake are var-elim patterns.*

Proof. Since it is known that BTP is a var-elim pattern [11], we only need to prove the result for the three existential patterns: ∃subBTP, ∃invsubBTP and ∃snake.

Every two-variable arc-consistent CSP instance allows either variable to be eliminated. So we only have to prove that these patterns allow variable elimination in CSP instances with at least three variables.

We first set up some general machinery which will be used in each of the three cases. Consider an arc-consistent CSP instance $I = \langle X, D, A, \text{cpt}\rangle$ and let s be a partial solution on $X \setminus \{x\}$.

Fix some assignment $\langle x, d\rangle$, and let:

$$Y = \{y \in X \setminus \{x\} \mid \text{cpt}(\langle y, s(y)\rangle, \langle x, d\rangle) = \texttt{TRUE}\},$$

$$\overline{Y} = \{z \in X \setminus \{x\} \mid \text{cpt}(\langle z, s(z)\rangle, \langle x, d\rangle) = \texttt{FALSE}\}.$$

For all $y, z \in X \setminus \{x\}$, since s is a partial solution, $\text{cpt}(\langle y, s(y)\rangle, \langle z, s(z)\rangle) = \texttt{TRUE}$. Thus, if $X = Y \cup \{x\}$ then we can extend s to a solution to I by choosing value d for variable x. So, in this case x could be eliminated. So we assume from now on that $\overline{Y} \neq \emptyset$.

By arc consistency, for all $z \in \overline{Y}$, there is some $\langle z, t(z)\rangle \in A$ such that $\text{cpt}(\langle z, t(z)\rangle, \langle x, d\rangle) = \texttt{TRUE}$.

We now prove the result for each pattern in turn.

Suppose that ∃subBTP does not occur at x in I for the mapping $a \mapsto d$. Consider any $y \in \overline{Y}$. By arc consistency, $\exists b \in \mathcal{D}(x)$ such that $\text{cpt}(\langle y, s(y)\rangle, \langle x, b\rangle) = \texttt{TRUE}$. Since the pattern ∃subBTP does not occur, and in particular on the set of assignments $\{\langle y, s(y)\rangle, \langle z, s(z)\rangle, \langle x, d\rangle, \langle x, b\rangle\}$, we can deduce that, for every variable $z \in X$ different from both x and y, $\text{cpt}(\langle z, s(z)\rangle, \langle x, b\rangle) = \texttt{TRUE}$. Hence, we can extend s to a solution to I by choosing $s(x) = b$. So, in any case x can be eliminated and ∃subBTP is indeed a var-elim pattern.

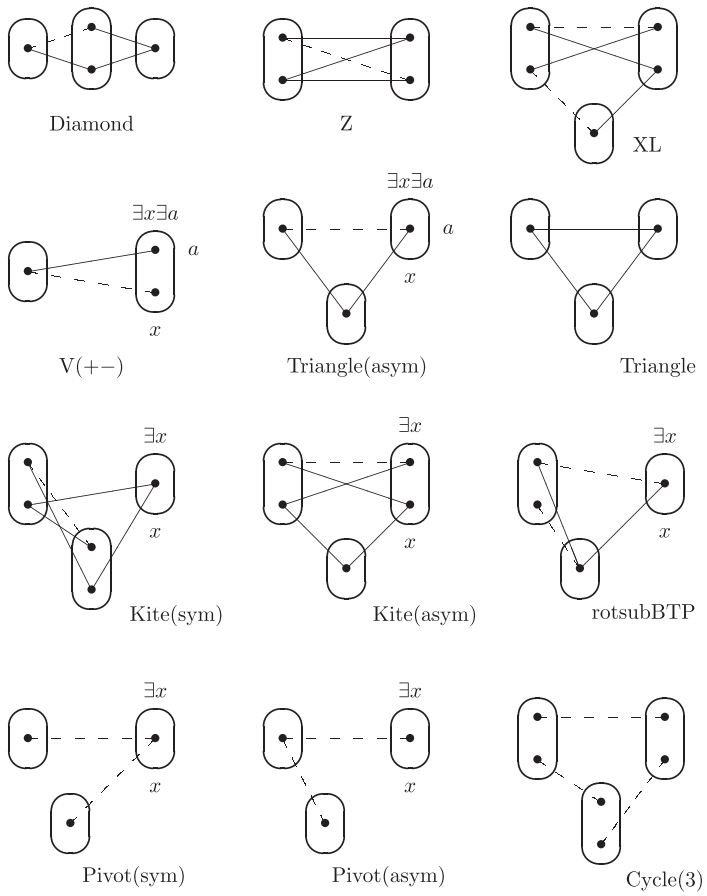

Fig. 3. Patterns which do not allow variable elimination.

Now instead, suppose ∃invsubBTP does not occur at x in I for the mapping $a \mapsto d$. Since the pattern ∃invsubBTP does not occur, if both y and z belong to $\overline{Y}$ then $\mathrm{cpt}(\langle y, t(y)\rangle, \langle z, t(z)\rangle) = \mathtt{TRUE}$, otherwise the pattern would occur on the assignments $\{\langle y, s(y)\rangle, \langle y, t(y)\rangle, \langle z, t(z)\rangle, \langle x, d\rangle\}$. Also, if $y \in Y$, $z \in \overline{Y}$, then $\mathrm{cpt}(\langle y, s(y)\rangle, \langle z, t(z)\rangle) = \mathtt{TRUE}$, otherwise the pattern would occur on $\{\langle z, s(z)\rangle, \langle z, t(z)\rangle, \langle y, s(y)\rangle, \langle x, d\rangle\}$.

So, in this case we have a solution s' to I, where

$$s'(v) = \begin{cases} d & \text{if } v = x, \\ s(v) & \text{if } v \in Y, \\ t(v) & \text{otherwise.} \end{cases}$$

So ∃invsubBTP is indeed a var-elim pattern.

For the final pattern, suppose that ∃snake does not occur at x in I for the mapping $a \mapsto d$. If $y \in Y$, $z \in \overline{Y}$, since the pattern ∃snake does not occur, we can deduce that $\mathrm{cpt}(\langle y, s(y)\rangle, \langle z, t(z)\rangle) = \mathtt{TRUE}$, otherwise the pattern would occur on the assignments $\{\langle z, s(z)\rangle, \langle z, t(z)\rangle, \langle y, s(y)\rangle, \langle x, d\rangle\}$. If both y and z both belong to $\overline{Y}$, then we can deduce first that $\mathrm{cpt}(\langle y, s(y)\rangle, \langle z, t(z)\rangle) = \mathtt{TRUE}$ (as in the previous case) and then, as a consequence, that $\mathrm{cpt}(\langle y, t(y)\rangle, \langle z, t(z)\rangle) = \mathtt{TRUE}$ (otherwise the pattern would occur on $\{\langle y, s(y)\rangle, \langle y, t(y)\rangle, \langle z, t(z)\rangle, \langle x, d\rangle\}$).

So, again in this case we have a solution s' to I, where s' is defined as above. So ∃snake is also a var-elim pattern. □

4. Characterisation of quantified var-elim patterns

Our aim is to precisely characterise all irreducible patterns which allow variable elimination in an arc-consistent binary CSP instance. We begin by identifying many patterns, including all those shown in Fig. 3, which are not variable elimination patterns.

Lemma 4.1. *None of the following patterns allow variable elimination in arc-consistent binary CSP instances: any pattern on strictly more than three variables, any pattern with three non-mergeable values for the same variable, any pattern with two non-mergeable*

incompatibility edges in the same constraint, Diamond, Z, XL, V(+−), Triangle(asym), Triangle, Kite(sym), Kite(asym), rotsubBTP, Pivot(asym), Pivot(sym), Cycle(3).

Proof. For each pattern we exhibit a binary arc-consistent CSP instance that:

- has a partial solution on the set of all the variables except a specified variable x;
- has no solution;
- does not contain the given pattern P at variable x (if P is a quantified pattern) or does not contain P at any variable (if P is a non-quantified pattern).

By definition, any such instance is enough to prove that a pattern is not a var-elim pattern.

- For any pattern P which is either Diamond, Z, XL, or Triangle, or has at least four variables, or has three non-mergeable values for the same variable.
  Let I_3^{2COL} be the CSP instance (corresponding to 2-colouring on 3 variables) with three Boolean variables, where the constraint between any two variables forces them to take different values.
  This instance has partial solutions on any two variables, but has no solution, and does not contain P.
- For V(+−) and Triangle(asym).
  Let $I_4^{\exists}$ be the instance on four variables x_1, x_2, x_3 and x, where the domains of x_1, x_2 and x_3 are all $\{0,1,2\}$ and the domain of x is $\{0,1,2,3\}$. Each pair of variables in $\{x_1, x_2, x_3\}$ must take values in $\{\langle 0,0\rangle, \langle 1,2\rangle, \langle 2,1\rangle\}$. There are three further constraints: for $i = 1,2,3$, we have that $(x_i > 0) \vee (x = i)$.
  $I_4^{\exists}$ has a partial solution on $\{x_1, x_2, x_3\}$ but has no solution. $I_4^{\exists}$ contains neither V(+−) nor Triangle(asym) at variable x for the value mapping $m(a) = 0$.
- For Kite(sym).
  Let I_4 be the CSP instance on four variables x_1, x_2, x_3, x where x_1, x_2 and x_3 are Boolean and $\mathcal{D}(x) = \{1,2,3\}$, with the following constraints: $x_1 \vee x_2$, $x_1 \vee x_3$, $x_2 \vee x_3$, $x_i \Leftrightarrow (x = i)$ $(i = 1,2,3)$.
  I_4 has a partial solution on $\{x_1, x_2, x_3\}$, has no solution, and does not contain Kite(sym) at variable x.
- For Kite(asym).
  Let I_4^{ZOA} be the CSP instance on the four variables x_1, x_2, x_3, x each with domain $\{1,2,3\}$, with the following constraints: $x_1 = x_2$, $x_1 = x_3$, $x_2 = x_3$, $(x_1 = 1) \vee (x = 1)$, $(x_2 = 2) \vee (x = 2)$, $(x_3 = 3) \vee (x = 3)$.
- For rotsubBTP.
  Define the three binary relations:

$$R = \{\langle 0,0\rangle, \langle 1,2\rangle, \langle 2,1\rangle\},$$
$$R_0 = \{\langle 0,0\rangle, \langle 1,1\rangle, \langle 2,1\rangle\},$$
$$R_1 = \{\langle 0,1\rangle, \langle 1,0\rangle, \langle 2,0\rangle\}.$$

  Let I_7 be the CSP instance on the seven variables $x_1, \ldots, x_6, x$ where $\mathcal{D}(x_i) = \{0,1,2\}$, for $i = 1, \ldots, 6$, and $\mathcal{D}(x) = \{0,1\}$, with the following constraints:
  For $(1 \le i < j \le 3)$ and $(4 \le i < j \le 6)$, $\langle x_i, x_j\rangle$ must take values in R.
  For $(1 \le i \le 3)$, $\langle x_i, x\rangle$ must take values in R_0.
  For $(4 \le i \le 6)$, $\langle x_i, x\rangle$ must take values in R_1.
- For the pattern Pivot(sym).
  Let I_4^{SAT} be the 2SAT instance on four Boolean variables x_1, x_2, x_3, x with the following constraints: $x_1 \equiv x_2$, $x_1 \equiv x_3$, $x_2 \vee x_3$, $\overline{x_2} \vee x$, $\overline{x_3} \vee \bar{x}$.
- For Cycle(3) or Pivot(asym), or any pattern with two non-mergeable incompatibility edges in the same constraint.
  Let I_6^{SAT} be the 2SAT instance on six Boolean variables $x_1, x_2, x_3, x_4, x_5, x$ with the following constraints: $\overline{x_1} \vee \overline{x_2}$, $\overline{x_1} \vee \overline{x_4}$, $x_1 \vee \overline{x_3}$, $x_1 \vee \overline{x_5}$, $x_2 \vee \bar{x}$, $x_4 \vee x$, $x_3 \vee \bar{x}$, $x_5 \vee x$. □

The following lemma is then key to proving that we have identified all possible irreducible quantified var-elim patterns.

Lemma 4.2. *The only flat quantified irreducible patterns that do not contain any of the patterns listed in Lemma 4.1 are contained in BTP, invsubBTP or snake (shown in Fig. 2).*

Proof. Consider a flat quantified irreducible pattern $P = \langle X, D, A, \text{cpt}\rangle$ that does not contain any of the patterns listed in Lemma 4.1. Thus P has at most three variables, each with domain size at most two.

We consider first the case of a 2-variable pattern P. By Lemma 4.1, P does not have two non-mergeable incompatibility edges and does not contain Z. Since P is irreducible and hence does not have any dangling assignment, we can deduce by exhausting over all possibilities that P does not have any compatibility edge and a single incompatibility edge. Hence P is contained in BTP. We can therefore assume that P has exactly three variables.

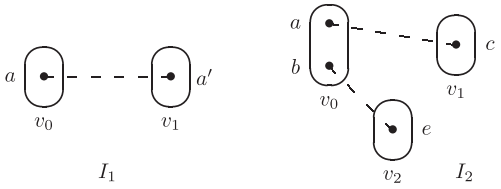

Fig. 4. The possible negative skeletons of var-elim patterns.

Now consider the negative sub-pattern $P^- = \langle X, D, A, \text{neg}\rangle$ where the compatibility function neg is cpt with its domain reduced to the incompatible pairs of assignments of P.

Any irreducible pattern on three variables that does not contain an incompatible pair of assignments must contain Triangle. Moreover, if any assignment is incompatible with two other assignments then P must contain either Pivot(sym) or Pivot(asym), or have two non-mergeable incompatible edges in the same constraint. Now, since P does not contain Cycle(3), it follows that P^- is I_1 or I_2, as shown in Fig. 4.

We first consider the latter case. Without loss of generality, we assume that b is compatible with c, to avoid a and b being mergeable.

Since the domains have at most two elements, we begin by assuming that $\mathcal{D}(v_1) = \{c, d\}$ and $\mathcal{D}(v_2) = \{e, f\}$. In this case a and d must be compatible to avoid d and c being mergeable. Also b and f must be compatible to stop e and f being mergeable. Now d and b cannot be compatible since otherwise Z occurs in P. Moreover, d and e cannot be compatible since otherwise XL occurs in P. Furthermore, d and f cannot be compatible since, whichever variable is chosen for $\overline{v}(P)$, either Kite(sym) or Kyte(asym) occurs in P. It follows that d can be removed as it is a dangling assignment.

Now we begin again. As before, to avoid e and f being mergeable or Diamond occurring in P, we have that f is compatible with b and not compatible with a. To avoid Triangle occurring in P, f cannot be compatible with c, which means that f can be removed since it is a dangling assignment.

So, we have $\mathcal{D}(v_1) = \{c\}$ and $\mathcal{D}(v_2) = \{e\}$. Suppose that there is a compatibility edge between c and e. If the distinguished variable $\overline{v}(P)$ is v_0 then, whether or not there is a compatibility edge between a and e, the pattern is contained in BTP. If $\overline{v}(P) = v_1$ and there is no compatibility edge between a and e, then the pattern is contained in invsubBTP. If $\overline{v}(P) = v_1$ and there is a compatibility edge between a and e, then the pattern contains rotsubBTP. If $\overline{v}(P) = v_2$, then the pattern contains rotsubBTP. Since we have covered all cases in which there is a compatibility edge between c and e, we assume that there is no edge between c and e.

Whether or not there is an incompatibility edge between a and e, the pattern is contained in BTP if $\overline{v}(P) = v_0$, and the pattern is contained in snake if $\overline{v}(P)$ is either v_1 or v_2.

The final case to consider is when P is a 3-variable pattern with $P^- = I_1$. Any two assignments for the third variable v_2 could be merged, so we can assume its domain is a singleton which we denote by $\{a''\}$. Since P is irreducible, does not contain Diamond, Z, Triangle, Kite(sym) or Kite(asym), we can deduce that the only compatible pairs of assignments include a''. In fact, both $\{a, a''\}$ and $\{a', a''\}$ must be compatible since P is irreducible. But then P is contained in BTP if $\overline{v}(P)$ is either v_0 or v_1, and is contained in invsubBTP if $\overline{v}(P) = v_2$. □

We need the following technical lemma which shortens several proofs.

Lemma 4.3. *If a pattern P occurs in a var-elim pattern Q with $|e(Q)| \leq 1$, then P is also a var-elim pattern.*

Proof. Suppose that P occurs in the var-elim pattern Q and that $|e(Q)| \leq 1$. By transitivity of the occurrence relation, if Q occurs in a binary CSP instance I (at variable x), then so does P. It follows that if (there is an injective mapping $m : e(P) \to \mathcal{D}(x)$ for which) P does not occur (at variable x) in an arc consistent binary CSP instance I, then (there is an injective mapping $m' : e(Q) \to \mathcal{D}(x)$ for which) Q does not occur (at variable x) and hence variable elimination is possible. □

The condition $|e(Q)| \leq 1$ is required in the statement of Lemma 4.3, since for an instance in which $\mathcal{D}(x)$ is a singleton, if $|e(P)| \leq 1$ and $|e(Q)| > 1$ there may be an injective mapping $m : e(P) \to \mathcal{D}(x)$ for which P does not occur at x but there can clearly be no injective mapping $m' : e(Q) \to \mathcal{D}(x)$.

According to Definition 2.7, a flat quantified pattern P is a sub-pattern of any existential version Q of P (and hence P occurs in Q). We state this special case of Lemma 4.3 as a corollary.

Corollary 4.1. *Let Q be an existential var-elim pattern with $|e(Q)| = 1$. If P is the flattened version of pattern Q, corresponding to $e(P) = \emptyset$, then P is also a var-elim pattern.*

The following lemma deals with the case of existential patterns P with $|e(P)| > 1$.

Lemma 4.4. *No irreducible existential pattern P with $|e(P)| > 1$ is a var-elim pattern.*

Proof. Let a_1, a_2 be two distinct assignments in $e(P)$. Since P is irreducible, a_1 and a_2 are not mergeable; so there is an assignment b such that $\langle b, a_1\rangle$ is a compatibility edge and $\langle b, a_2\rangle$ is an incompatibility edge (or vice versa) in P.

Consider the instance I_4^k (where $k = |e(P)| + 3$) on four variables x_1, x_2, x_3, x with domains $\mathcal{D}(x_1) = \mathcal{D}(x_2) = \mathcal{D}(x_3) = \{0, 1, 2\}$, $\mathcal{D}(x) = \{1, \ldots, k\}$ and the following constraints: $x_1 = 2 - x_2$, $x_1 = 2 - x_3$, $x_2 = 2 - x_3$, $(x_i \neq 1) \vee (x = i)$ $(i = 1, 2, 3)$. I_4^k has a partial solution (1,1,1) on variables x_1, x_2, x_3 but has no solution. Furthermore, for any (arbitrary choice of) injective mapping $m : e(P) \to \mathcal{D}(x)$ which maps $e(P)$ to a subset of $\{4, \ldots, k\}$, P does not occur on x since the values $m(a_1), m(a_2) \in \{4, \ldots, k\}$ have the same compatibilities with all assignments to other variables in I_4^k.

Therefore there are no irreducible var-elim patterns P with $|e(P)| > 1$. □

The following theorem is a direct consequence of Theorem 3.1 and Corollary 4.1 together with Lemma 4.1, Lemma 4.2 and Lemma 4.3.

Theorem 4.1. *The irreducible flat quantified patterns allowing variable elimination in arc-consistent binary CSP instances are BTP, invsubBTP or snake (and their irreducible sub-patterns).*

We are now able to provide the characterisation for existential patterns after a little extra work.

Theorem 4.2. *The only irreducible existential patterns which allow variable elimination in arc-consistent binary CSP instances are ∃subBTP, ∃invsubBTP, ∃snake (and their irreducible sub-patterns).*

Proof. By Lemma 4.4 we only need to consider patterns P with $|e(P)| = 1$.

We know from Theorem 3.1 that ∃subBTP, ∃invsubBTP, ∃snake are var-elim patterns.

Theorem 4.1 and Corollary 4.1 show that when we flatten an existential var-elim pattern then the resulting flat quantified pattern is contained in BTP, invsubBTP or snake.

In the case of invsubBTP and snake, the existential versions of these patterns are var-elim patterns and so there is nothing left to prove. So we only need to consider quantified patterns which flatten into sub-patterns of BTP.

Let ∃BTP denote the existential version Q of BTP such that $|e(Q)| = 1$. By symmetry, ∃BTP is unique. The only remaining case is when P is an irreducible sub-pattern of ∃BTP with $|e(P)| = 1$. By a straightforward exhaustive case analysis, we find that, in this case, either P contains V(+−) or Triangle(asym) or P is a sub-pattern of ∃subBTP. The result then follows by Lemma 4.1 and Lemma 4.3. □

Combining Theorem 4.1 and Theorem 4.2, we obtain the characterisation of irreducible quantified var-elim patterns.

Theorem 4.3. *The only irreducible quantified patterns which allow variable elimination in arc-consistent binary CSP instances are BTP, ∃subBTP, ∃invsubBTP, ∃snake (and their irreducible sub-patterns).*

It is easy to see that variable elimination cannot destroy arc consistency. Hence there is no need to re-establish arc consistency after variable eliminations. Furthermore, the result of applying our var-elim rules until convergence is unique; variable eliminations may lead to new variable eliminations but cannot introduce patterns and hence cannot invalidate applications of our var-elim rules.

5. Value elimination patterns

We now consider when forbidding a pattern can allow the elimination of values from domains rather than the elimination of variables. Value-elimination is at the heart of the simplification operations employed by constraint solvers during preprocessing or during search. In current solvers such eliminations are based almost exclusively on consistency operations: a value is eliminated from the domain of a variable if this assignment can be shown to be inconsistent (in the sense that it cannot be part of any solution). Another value-elimination operation which can be applied is neighbourhood substitutability which allows the elimination of certain assignments which are unnecessary for determining the satisfiability of the instance. Neighbourhood substitutability can be described by means of the pattern shown in Fig. 5. If in a binary CSP instance I, there are two assignments a, b for the same variable x such that this pattern does not occur (meaning that a is consistent with all assignments with which b is consistent), then the assignment b can be eliminated. This is because in any solution containing b, simply replacing b by a produces another solution.

It is worth noting that even when all solutions are required, neighbourhood substitutability can still be applied since all solutions to the original instance can be recovered from the set of solutions to the reduced instance in time which is linear in the total number of solutions and polynomial in the size of the instance [14].

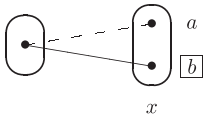

Fig. 5. A value elimination pattern corresponding to neighbourhood substitution.

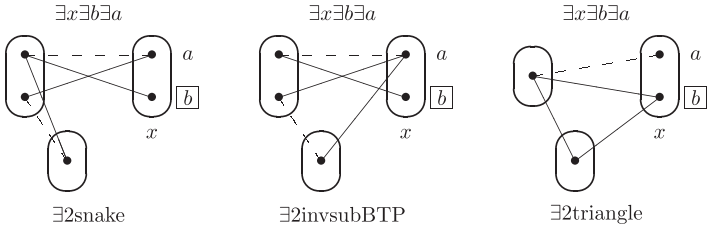

Fig. 6. Three val-elim patterns.

Definition 5.1. We say that a value $b \in \mathcal{D}(x)$ *can be eliminated* from an instance I if the instance I' in which the assignment b has been deleted from $\mathcal{D}(x)$ is satisfiable if and only if I is satisfiable.

Definition 5.2. An existential pattern P with a distinguished value $\overline{val}(P)$ is a *value elimination pattern (val-elim pattern)* if in all arc-consistent instances I, whenever the pattern does not occur at a variable x in I for at least one injective value mapping m, the value $m(\overline{val}(P))$ can be eliminated from $\mathcal{D}(x)$ in I.

An obvious question is which patterns allow value elimination while preserving satisfiability? The following theorem gives three existential patterns which provide strict generalisations of neighbourhood substitutability since in each case the pattern of Fig. 5 is a sub-pattern. In each of the patterns P in Fig. 5 and Fig. 6, the value that can be eliminated $\overline{val}(P)$ is the value b surrounded by a small box.

Theorem 5.1. *The existential patterns shown in Fig. 6, namely ∃2snake, ∃2invsubBTP and ∃2triangle, are each val-elim patterns.*

Proof. We first show that the result holds for instances I with at most two variables. Let x be a variable of I. For $|\mathcal{D}(x)| > 1$, there is clearly an injective mapping $m : e(P) \to \mathcal{D}(x)$ for which none of the patterns P shown in Fig. 6 occur since they all have three variables. But, we can always eliminate all but one value in $\mathcal{D}(x)$ without destroying satisfiability, since by arc consistency the remaining value is necessarily part of a solution. For $|\mathcal{D}(x)| \leq 1$, there can be no injective mapping $m : e(P) \to \mathcal{D}(x)$ since $|e(P) = 2|$ and hence there is nothing to prove. In the rest of the proof we therefore only need to consider instances $I = \langle X, D, A, \text{cpt}\rangle$ with at least three variables. We will prove the result for each of the three patterns one by one.

We consider first ∃2snake. Suppose that for a variable x and values $a, b \in \mathcal{D}(x)$, the pattern ∃2snake does not occur. Let I' be identical to I except that value b has been eliminated from $\mathcal{D}(x)$. Suppose that s is a solution to I with $s(x) = b$. It suffices to show that I' has a solution. Let Y ($\overline{Y}$) be the set of variables $z \in X \setminus \{x\}$ such that $\text{cpt}(\langle z, s(z)\rangle, \langle x, a\rangle) = \texttt{TRUE}$ ($\texttt{FALSE}$). By arc consistency, there are assignments $\langle z, t(z)\rangle$ for all $z \in \overline{Y}$ which are compatible with $\langle x, a\rangle$. Let $z \in \overline{Y}$ and $y \in X \setminus \{x, z\}$. Since s is a solution with $s(x) = b$, $\text{cpt}(\langle y, s(y)\rangle, \langle z, s(z)\rangle) = \text{cpt}(\langle x, b\rangle, \langle z, s(z)\rangle) = \texttt{TRUE}$. Since ∃2snake does not occur on $\{\langle x, a\rangle, \langle x, b\rangle, \langle z, s(z)\rangle, \langle z, t(z)\rangle, \langle y, s(y)\rangle\}$, we can deduce that $\text{cpt}(\langle z, t(z)\rangle, \langle y, s(y)\rangle) = \texttt{TRUE}$. In particular, we have $\text{cpt}(\langle y, s(y)\rangle, \langle z, t(z)\rangle) = \texttt{TRUE}$ for all $y \neq z \in \overline{Y}$. Then, since ∃2snake does not occur on the assignments $\{\langle x, a\rangle, \langle x, b\rangle, \langle y, s(y)\rangle, \langle y, t(y)\rangle, \langle z, t(z)\rangle\}$, we can deduce $\text{cpt}(\langle z, t(z)\rangle, \langle y, t(y)\rangle) = \texttt{TRUE}$. Hence the assignments $\langle z, t(z)\rangle$ ($z \in \overline{Y}$) are compatible between themselves, are all compatible with all $\langle y, s(y)\rangle$ ($y \in Y$) and with $\langle x, a\rangle$. Thus, s' is a solution to I', where

$$s'(v) = \begin{cases} a & \text{if } v = x, \\ s(v) & \text{if } v \in Y, \\ t(v) & \text{otherwise.} \end{cases}$$

We now consider ∃2invsubBTP. Suppose that for a variable x and values $a, b \in \mathcal{D}(x)$ in an instance I, the pattern ∃2invsubBTP does not occur. Let I' be identical to I except that value b has been eliminated from $\mathcal{D}(x)$. Suppose that s is a solution to I with $s(x) = b$ and again let Y ($\overline{Y}$) be the set of variables $z \in X \setminus \{x\}$ such that $\text{cpt}(\langle z, s(z)\rangle, \langle x, a\rangle) = \texttt{TRUE}$ ($\texttt{FALSE}$). By arc consistency, for each $z \in \overline{Y}$, there is an assignment $\langle z, t(z)\rangle$ which is compatible with $\langle x, a\rangle$. Let s' be defined as above. Consider $v \in X \setminus \{x\}$. We know that $\text{cpt}(\langle x, a\rangle, \langle v, s'(v)\rangle) = \texttt{TRUE}$. Let $z \in \overline{Y}$. Since the pattern ∃2invsubBTP does not occur on $\{\langle x, a\rangle, \langle x, b\rangle, \langle z, s(z)\rangle, \langle z, t(z)\rangle, \langle v, s'(v)\rangle\}$, we can deduce that $\text{cpt}(\langle z, t(z)\rangle, \langle v, s'(v)\rangle) = \texttt{TRUE}$. It follows that s' is a solution to I'.

Finally, we consider ∃2triangle. Suppose that in an instance I, for values $a, b \in \mathcal{D}(x)$, the pattern ∃2triangle does not occur. Let I' be identical to I except that value b has been eliminated from $\mathcal{D}(x)$. Suppose that s is a solution to I with $s(x) = b$. Then $\langle x, a\rangle$ must be compatible with all assignments $\langle y, s(y)\rangle$ ($y \in X \setminus \{x\}$), otherwise the pattern ∃2triangle would occur on $\{\langle x, a\rangle, \langle x, b\rangle, \langle y, s(y)\rangle, \langle z, s(z)\rangle\}$ for all $z \in X \setminus \{x, y\}$. It follows that s'' is a solution to I', where

$$s''(v) = \begin{cases} a & \text{if } v = x, \\ s(v) & \text{otherwise.} \end{cases} \qquad \square$$

Example 5.1. Consider a CSP instance corresponding to a problem of colouring a complete graph on four vertices. The colours assigned to the four vertices are represented by variables x_1, x_2, x_3, x_4 whose domains are, respectively, $\{0, 1, 2, 3\}$, $\{0, 1\}$, $\{0, 2\}$, $\{0, 3\}$. Notice that the instance is arc consistent and no eliminations are possible by neighbourhood substitution. However, the value 1 can be eliminated from the domain of x_1 since for the mapping $a \mapsto 0$, $b \mapsto 1$, the pattern ∃2snake does not occur on x_1. The values 2 and 3 can also be eliminated from the domain of x_1 for the same reason. After applying arc consistency to the resulting instance, all domains are singletons.

Example 5.2. Consider the arc-consistent instance on three Boolean variables x, y, z and with the constraints $z \vee \neg x$, $z \vee y$, $\neg y \vee \neg x$. In this instance we can eliminate the assignment $\langle x, 0\rangle$ since ∃2invsubBTP does not occur on variable x for the mapping $a \mapsto 1$, $b \mapsto 0$. The assignments $\langle y, 1\rangle$ and $\langle z, 0\rangle$ then have no support at x and hence can be eliminated by arc consistency, leaving an instance in which all domains are singletons.

Example 5.3. Consider the arc-consistent CSP instance corresponding to a graph colouring problem on a complete graph on three vertices in which the domains of variables x_1, x_2, x_3 are each $\{0, 1\}$. Again, no eliminations are possible by neighbourhood substitution. However, the value 1 can be eliminated from the domain of x_1 since for the mapping $a \mapsto 0$, $b \mapsto 1$, the pattern ∃2triangle does not occur on x_1. Applying arc consistency then leads to an empty domain from which we can deduce that the original instance was unsatisfiable.

Neighbourhood substitution cannot destroy arc consistency [14], but eliminating a value by a val-elim pattern can provoke new eliminations by arc consistency, as we have seen in the above examples.

The result of applying a sequence of neighbourhood substitution eliminations until convergence is unique modulo isomorphism [14]. This is not true for the result of eliminating domain elements by val-elim patterns, as the following example demonstrates.

Example 5.4. Consider the CSP instance on three variables x_1, x_2, x_3, each with domain $\{0, 1, 2\}$, and with the following constraints: $(x_1 \neq 2) \vee (x_2 \neq 2)$, $(x_1, x_3) \in R$, $(x_2, x_3) \in R$, where R is the relation $\{(0, 0), (0, 2), (1, 1), (2, 1), (2, 2)\}$. We can eliminate the assignment $\langle x_3, 0\rangle$ since ∃2snake does not occur on x_3 with the value a mapping to 2 and b to 0. But then in the resulting arc-consistent instance, no more eliminations are possible by any of the val-elim patterns shown in Fig. 6. However, in the original instance we could have eliminated the assignment $\langle x_3, 1\rangle$ since ∃2snake does not occur on x_3 with the value a mapping to 0 and b to 1. Then we can successively eliminate $\langle x_1, 1\rangle$, $\langle x_2, 1\rangle$ by arc consistency and then $\langle x_1, 2\rangle$, $\langle x_2, 2\rangle$, $\langle x_3, 0\rangle$ by ∃2snake. In the resulting instance all domains are singletons. Thus, for this instance there are two convergent sequences of value eliminations which produce non-isomorphic instances.

It is clear that variable elimination by our var-elim rules can provoke new value eliminations by our val-elim rules. Value elimination may provoke new variable eliminations, but may also invalidate a variable elimination if the value eliminated (or one of the values eliminated by subsequent arc consistency operations) is the only value on which an existential var-elim pattern does not occur. Thus, to maximise reductions, variable eliminations should always be performed before value eliminations.

6. Characterisation of value elimination patterns

As with existential variable-elimination patterns, we can give a dichotomy for irreducible existential val-elim patterns. We first require the following lemma which shows that many patterns, including those illustrated in Fig. 7 (along with the patterns Z and Diamond shown in Fig. 3), cannot be contained in val-elim patterns. In Fig. 7, each of the patterns I(−), L(+−), triangle1, triangle2, ∃Kite, ∃Kite(asym) and ∃Kite1 has a distinguished value $b = \overline{val}(P)$ which is highlighted in the figure by placing the value in a small box.

Lemma 6.1. *None of the following existential patterns P (with a distinguished value $\overline{val}(P)$) allow value elimination in arc-consistent binary CSP instances: any pattern on strictly more than three variables, any pattern with three non-mergeable values for the same variable $v \neq \overline{v}(P)$, any pattern with two non-mergeable incompatibility edges in the same constraint, any pattern containing any of Z, Diamond, I(−), L(−), L(+−), triangle1, triangle2, ∃Kite, ∃Kite(asym) or ∃Kite1.*

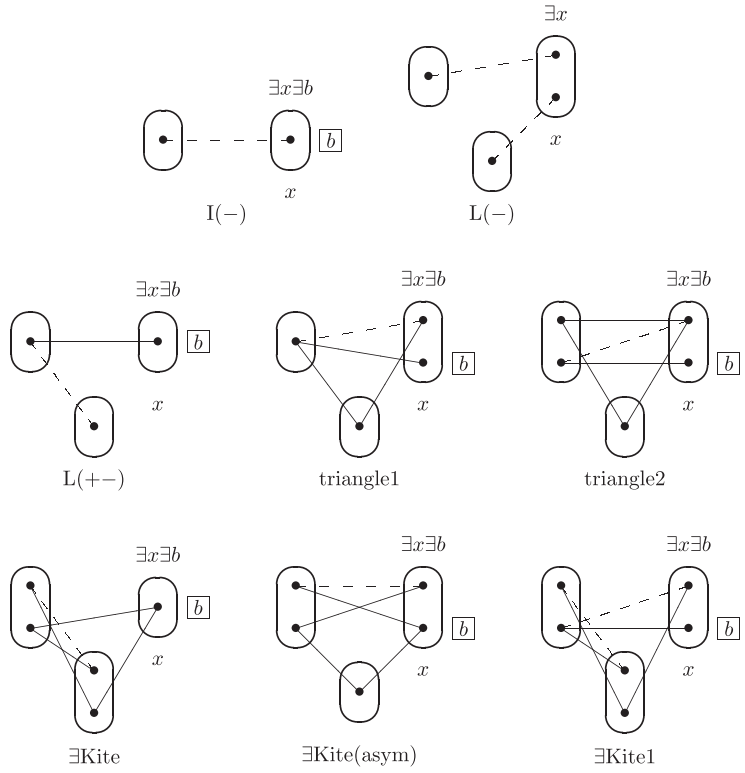

Fig. 7. Patterns which do not allow value elimination.

Proof. For each pattern we exhibit a binary arc-consistent CSP instance I and a value b for a variable x in I such that

- I has a solution which includes the assignment $\langle x, b\rangle$;
- I has no solution if the value b is deleted from $\mathcal{D}(x)$;
- I does not contain the given pattern P on x with $\overline{val}(P)$ mapping to b.

By definition, any such instance is sufficient to prove that the pattern P is not a val-elim pattern. Since in existential patterns P, the set $e(P)$ may be of arbitrary size, we have to give generic instances in which the size of the domain of x is arbitrarily large.

- For any pattern P which either contains I$(-)$ or has strictly more than three variables or with three non-mergeable values for the same variable $v \neq \overline{v}(P)$.
  Let I_3^{SAT} be the arc-consistent instance on three variables x_1, x_2, x with domains $\mathcal{D}(x_1) = \mathcal{D}(x_2) = \{0, 1\}$, $\mathcal{D}(x) = \{0, \ldots, k\}$ and with the following constraints: $\overline{x_1} \vee \overline{x_2}$, $x_1 \vee (x = 0)$, $x_2 \vee (x = 0)$. I_3^{SAT} has a solution $\langle 0, 0, 0\rangle$ which includes the assignment $\langle x, 0\rangle$, has no solution if this assignment is eliminated, and does not contain P on x with $\overline{val}(P)$ mapping to 0.
- For any pattern P which has two non-mergeable incompatibility edges in the same constraint.
  Let I_{2k+1}^{SAT} be the arc-consistent instance on $2k + 1$ variables $x_1, \ldots, x_{2k}, x$ with domains $\mathcal{D}(x_1) = \ldots = \mathcal{D}(x_{2k}) = \{0, 1\}$, $\mathcal{D}(x) = \{0, \ldots, k\}$ and with the following constraints for each $i = 1, \ldots, k$: $\overline{x_{2i-1}} \vee \overline{x_{2i}}$, $x_{2i-1} \vee (x \neq i)$, $x_{2i} \vee (x \neq i)$. I_{2k+1}^{SAT} has a solution $\langle 0, \ldots, 0\rangle$ which includes the assignment $\langle x, 0\rangle$, has no solution if this assignment is eliminated, and does not contain P on x with $\overline{val}(P)$ mapping to 0.
- For any pattern P which contains L$(+-)$, triangle2 or ∃Kite1.
  Let I_3 be the arc-consistent instance on three variables x_1, x_2, x with domains $\mathcal{D}(x_1) = \mathcal{D}(x_2) = \mathcal{D}(x) = \{0, \ldots, k\}$ and with the following constraints: $(x_1 = 0) \vee (x_2 = 0)$, $x_1 = x$, $x_2 = x$. I_3 has a solution $\langle 0, 0, 0\rangle$ which includes the assignment $\langle x, 0\rangle$, has no solution if this assignment is eliminated, and does not contain P on x with $\overline{val}(P)$ mapping to 0.
- For any pattern P which contains L$(-)$.
  Let I_{3+} be the arc-consistent instance on four variables x_1, x_2, x_3, x with domains $\mathcal{D}(x_1) = \mathcal{D}(x_2) = \mathcal{D}(x_3) = \mathcal{D}(x) = \{0, \ldots, k\}$ and with the following constraints: $(x_1 = 0) \vee (x_2 = 0)$, $x_1 = x_3$, $x_2 = x_3$, $x_3 = x$. I_{3+} has a solution $\langle 0, 0, 0, 0\rangle$ which includes the assignment $\langle x, 0\rangle$, has no solution if this assignment is eliminated, and does not contain P on x.

- For any pattern P which contains triangle1, ∃Kite, ∃Kite(asym), Diamond or Z.
  Let I_3^{2k} be the arc-consistent instance on three variables x_1, x_2, x each with domain $\{0, \ldots, 2k\}$ and with the following constraints: $x_1 = 2k - x_2$, $x_1 = x$, $x_2 = x$. I_3^{2k} has a solution $\langle k, k, k\rangle$ which includes the assignment $\langle x, k\rangle$, has no solution if this assignment is eliminated, and does not contain P on x with $\overline{val}(P)$ mapping to k. □

We can now characterise those irreducible existential patterns which allow value elimination and hence generalise neighbourhood substitution.

Theorem 6.1. *The only irreducible existential patterns which allow value elimination in arc-consistent binary CSP instances are ∃2snake, ∃2invsubBTP and ∃2triangle (and their irreducible sub-patterns).*

Proof. Let P be an irreducible existential pattern which allows value elimination in arc-consistent binary CSP instances. Since $\overline{val}(P)$ is necessarily defined in a val-elim pattern and belongs to $e(P)$, we need to consider three different cases:

1. $|e(P)| = 1$, and hence $e(P) = \{\overline{val}(P)\}$,
2. $|e(P)| = 2$,
3. $|e(P)| > 2$.

Case $|e(P)| = 1$: Consider the CSP instance I_2 consisting of only two variables x_1, x_2, each with a singleton domain $\{0\}$ together with the constraint $x_1 = x_2$. Trivially, the value 0 cannot be eliminated from the domain of x_1 without changing the satisfiability of the instance. Any existential pattern P with $|e(P)| = 1$ containing more than two variables or at least one incompatibility edge does not occur in I_2 on x_1 with $\overline{val}(P)$ mapping to 0, and hence cannot be a val-elim pattern. There is no 2-variable irreducible existential pattern which contains only compatibility edges. Therefore, the only irreducible val-elim pattern P with $|e(P)| = 1$ is the trivial pattern with no edges (which is a sub-pattern of ∃2snake, for example).

Case $|e(P)| = 2$: Let $e(P) = \{a, b\}$ where $b = \overline{val}(P)$. We know from Lemma 6.1 that I(−) cannot be contained in a val-elim pattern. We can therefore deduce that the assignment $\langle \overline{v}(P), b\rangle$ can only belong to compatibility edges in P. Since P is irreducible, we can deduce that P must contain the neighbourhood substitution pattern shown in Fig. 5, otherwise a and b could be merged. By Lemma 6.1, P does not contain more than three variables, does not contain L(−) or L(+−) and does not contain more than one incompatibility edge per constraint. It follows that P contains at most two incompatibility edges. The only extension of the neighbourhood substitution pattern (shown in Fig. 5) containing only one incompatibility edge and containing none of I(−), Z, Diamond, triangle1, triangle2 or ∃Kite(asym) is the val-elim pattern ∃2triangle. The only extensions of the neighbourhood substitution pattern containing exactly two incompatibility edges and containing none of I(−), L(−), L(+−), Z, Diamond, triangle1, ∃Kite or ∃Kite1 are the val-elim patterns ∃2snake and ∃2invsubBTP. Hence, the only irreducible val-elim patterns P with $|e(P)| = 2$ are ∃2snake, ∃2invsubBTP and ∃2triangle (and their irreducible sub-patterns).

Case $|e(P)| > 2$: Let a_1, a_2, a_3 be three distinct values in $e(P)$ and, for $i = 1, 2, 3$, let q_i denote the assignment $\langle \overline{v}(P), a_i\rangle$. Since P is irreducible, for all i, j such that $1 \leq i < j \leq 3$, a_i and a_j are not mergeable; so there is an assignment p_{ij} such that $\langle p_{ij}, q_i\rangle$ is a compatibility edge and $\langle p_{ij}, q_j\rangle$ is an incompatibility edge (or vice versa) in P. By Lemma 6.1, P has at most three variables, including $\overline{v}(P)$. So two of p_{12}, p_{13}, p_{23} are assignments to the same variable. Without loss of generality, suppose that p_{12}, p_{13} are assignments to the same variable $y \neq \overline{v}(P)$. By Lemma 6.1, P has at most one incompatibility edge in each constraint. It follows that $p_{12} = p_{13}$, with $\langle p_{12}, q_1\rangle$ an incompatibility edge and $\langle p_{12}, q_2\rangle$, $\langle p_{12}, q_3\rangle$ compatibility edges. It also follows that p_{23} must be an assignment to a distinct variable $z \notin \{y, \overline{v}(P)\}$. But then P contains Diamond on p_{12}, q_2, q_3, p_{23} and so, by Lemma 6.1, cannot be a val-elim pattern. Therefore there are no irreducible val-elim patterns P with $|e(P)| > 2$. □

7. Recovering one or all solutions after eliminations

The binary CSP has diverse applications. In some applications it is only the satisfiability of the instance which is of interest. For example, in optimal planning, to determine whether an action a among a set of available actions A is indispensable (i.e. that it is present in all solution-plans) we need to determine the satisfiability of a binary CSP representing the same planning problem using the set of actions $A \setminus \{a\}$ [10]. The variable and value elimination rules presented in this paper are directly applicable to such problems.

Nonetheless, in most applications, the final aim is to find one or all solutions. In many planning, scheduling and configuration problems, the aim is often to find just one solution which satisfies all the constraints. In other application areas, such as fault-diagnosis [32] or the interpretation of ambiguous pictures [13], it is important to find all solutions or a representation of all solutions from which it is possible to extract in polynomial time a solution satisfying certain criteria. For example, the on-line configuration of a product (such as a car) by a user can be rendered tractable by the off-line compilation of all

solutions into some appropriate compact form [1,2]. We will therefore study in this section whether it is possible to efficiently recover one or all solutions to a binary CSP instance after elimination of variables and/or values by our rules. We will show that the efficient recovery of one solution is always possible, but that only some of our rules allow the efficient recovery of all solutions.

The elimination of a variable cannot destroy arc consistency, but the elimination of a value may do so. Throughout this section we assume that the elimination of an assignment by applying a value-elimination rule is necessarily immediately followed by the re-establishment of arc consistency.

Proposition 7.1. *Let I be an arc-consistent binary CSP instance and let s be a solution to the instance obtained after applying a sequence σ of variable and value elimination operations (defined by irreducible quantified var-elim or val-elim patterns in the sense of Definition 2.8 and Definition 5.2). Then a solution to I can be found from (s, σ) in $O(cd)$ time, where c is the number of non-trivial constraints and d the maximum domain size in I.*

Proof. Since value elimination does not modify constraints, any solution for an instance obtained from I by value eliminations is also a solution to I. We therefore only need to consider the case of var-elim rules. We identified the irreducible quantified var-elim patterns in Theorem 4.3 as (irreducible sub-patterns of) BTP, ∃subBTP, ∃invsubBTP or ∃snake. We only need to prove the proposition for these four patterns since absence of any sub-pattern implies absence of the pattern itself.

Consider a single variable elimination operation consisting in eliminating variable x. Let c_x denote the number of constraints whose scope includes x. If x has been eliminated due to the absence of BTP, then it is known that any solution s of the reduced instance can be extended to a solution of I [11]. The proof of Theorem 3.1 showed that this is also true in the case that x has been eliminated due to the absence of ∃subBTP. In order to extend s to a solution of I, it suffices to test each of the elements of $\mathcal{D}(x)$ in turn against each of the c_x constraints. This can be achieved in $O(c_x d)$ time.

For the two other patterns (∃invsubBTP and ∃snake), the proof of Theorem 3.1 actually provides an algorithm to recover a solution s' of I from the solution s of the reduced instance, via the calculation of the variable set $\overline{Y}$ and the assignments $t(z)$ $(z \in \overline{Y})$. Again this can be achieved in $O(c_x d)$ time.

Summing the $O(c_x d)$ complexity of recovering a solution to the instance in which a variable x is reinstated, over all eliminated variables x, gives a total complexity of $O(cd)$, as claimed. □

Proposition 7.2. *Let I be an arc-consistent binary CSP instance and let S be the set of all solutions to the instance obtained after applying a sequence σ of operations given by the var-elim patterns BTP, ∃subBTP and the val-elim pattern ∃2triangle. Then the set of all solutions to I can be found from (S, σ) in $O(|S_I|cd + 1)$ time, where S_I is the set of solutions to I.*

Proof. In the trivial case in which $|S_I| = 0$, we necessarily have as input $S = \emptyset$ which can clearly be tested for in $O(1)$ time. In another simple case, in which I has at most two variables, the result follows from arc consistency. We therefore only need to consider satisfiable instances with at least three variables.

We now consider the elimination of a single variable x from an instance I due to absence of one of the var-elim patterns BTP or ∃subBTP. As observed in the proof of Proposition 7.1, each solution of the reduced instance can be extended to a solution of I. This implies that the number of solutions cannot decrease when we reinstate the variable x. Clearly each solution of I is an extension of a solution of the reduced instance. So the algorithm given in the proof of Proposition 7.1, applied in turn to each solution of the reduced instance will find all solutions of I in time $O(|S_I|c_x d)$.

Now consider the elimination of a value b from the domain of a variable x by absence of ∃2triangle on values $a, b \in \mathcal{D}(x)$ in an instance I. As observed in the proof of Theorem 5.1, s is a solution to I with $s(x) = b$ implies that s'' defined by $s''(x) = a$, $s''(v) = s(v)$ for $v \neq x$ is a solution to the reduced instance. To determine all solutions of I from the set of all solutions of the reduced instance thus requires only $O(|S_I|c_x)$ time.

Summing over all variables x (and, in the case of value-eliminations, over all assignments to x), we obtain a total time complexity of $O(|S_I|cd + 1)$, as claimed. □

An essential element of the proof of Proposition 7.2 in the case of var-elim patterns is that the number of solutions does not decrease when a variable x is reinstated. Unfortunately, in the case of the var-elim patterns ∃invsubBTP and ∃snake, it is easy to construct an example in which this is not true. Indeed, consider a binary CSP instance I corresponding to the 2-colouring of a star graph (a graph composed of one central node with edges to $n-1$ other nodes). Let x be the variable corresponding to the central node of the graph. Since neither ∃invsubBTP nor ∃snake occur on x, both rules allow us to eliminate x, leaving an instance on $n-1$ variables and no constraints. Whereas I has only two solutions (corresponding to the two possible 2-colourings of a star graph), the reduced instance has 2^{n-1} solutions. In this example, reinstating a single variable decreased the number of solutions by an exponential factor.

The elimination of values can, on the other hand, dramatically simplify an instance to the extent that finding all solutions to the original instance remains intractable, as illustrated by the following proposition.

Proposition 7.3. *Let I be an arc-consistent binary CSP instance and suppose that we are given the set of all solutions to the instance obtained after applying a single value-elimination operation due to the absence of one of the patterns ∃2snake or ∃2invsubBTP. Determining whether I has more than one solution is NP-complete.*

Proof. The problem is clearly in NP. It therefore suffices to give a polynomial reduction from the known NP-complete problem binary CSP. Let $J = \langle X, D, A, \text{cpt}\rangle$ be an arbitrary instance of binary CSP. We will build an instance I_J such that, after elimination of one variable x from I_J by either ∃2invsubBTP or ∃2snake and re-establishing arc consistency, we obtain a trivially-solvable instance with exactly one solution, but determining the existence of a second solution to I_J is equivalent to solving the instance J.

The variable-set of instance I_J is $X \cup \{x\}$ (where x is a variable not in X). For each variable $y \in X$, the domain of y in I_J is $\mathcal{D}(y) \cup \{0\}$, where without loss of generality we assume that 0 does not belong to the domain $\mathcal{D}(y)$ of variable y in J. The domain of variable x in I_J is $\{0, 1\}$. The compatibility function of I_J is an extension of the compatibility function of J: for each $y \in X$, the assignment $\langle x, 0\rangle$ is compatible only with the assignment $\langle y, 0\rangle$, whereas the assignment $\langle x, 1\rangle$ is compatible with all the assignments $\langle y, a\rangle$ for $a \neq 0$; furthermore for each $y, z \in X$, the assignment $\langle y, 0\rangle$ is compatible with all assignments to z.

Neither ∃2invsubBTP nor ∃2snake occur on variable x in I_J with a, b mapping respectively to $0, 1$. We can therefore eliminate the value 1 from the domain of x. After establishing arc consistency, all domains are reduced to the singleton $\{0\}$. Hence the reduced instance has exactly one solution. In the instance I_J, the assignment $\langle x, 0\rangle$ only belongs to the solution assigning 0 to each variable, whereas the assignment $\langle x, 1\rangle$ is compatible with exactly the set of solutions to the instance J. Therefore, determining the existence of a second solution to I_J is equivalent to determining the satisfiability of J. □

8. Conclusion

This paper has introduced the notion of variable and value elimination rules in binary CSPs based on the absence of quantified patterns. We have identified all irreducible quantified patterns whose absence allows variable or value elimination. As a consequence, we have also identified novel tractable classes of binary CSPs. From a practical point of view, our rules can be incorporated into generic constraint solvers to prune the search tree.

There are several interesting directions for further research. Can we generalise the variable or value elimination patterns described in this paper to arbitrary-arity CSP instances (perhaps using one of the possible definitions of microstructure for constraints of arbitrary arity [25])? A partial positive answer to this question has recently been provided by arbitrary-arity versions of BTP [12]. Do the variable and value elimination patterns introduced in this paper generalise to other versions of constraint satisfaction, such as the QCSP (as is the case for the tractable class defined by BTP [19]) or the Weighted CSP (as is the case for tractable class defined by the so-called joint-winner pattern [9])? The research reported in the present paper has recently led to the discovery of sound variable and value elimination rules defined by local properties which strictly generalise the absence of patterns [15]. The characterisation of all such generalised variable or value elimination rules is a challenging open problem.

Acknowledgments

Martin Cooper and Guillaume Escamocher were supported by ANR Project ANR-10-BLAN-0210. Stanislav Živný was supported by a Royal Society University Research Fellowship. David Cohen, Martin Cooper and Stanislav Živný were supported by EPSRC grant EP/L021226/1.

References

[1] Jérôme Amilhastre, Hélène Fargier, Pierre Marquis, Consistency restoration and explanations in dynamic CSPs application to configuration, Artif. Intell. 135 (1–2) (2002) 199–234.
[2] Jérôme Amilhastre, Hélène Fargier, Alexandre Niveau, Cédric Pralet, Compiling CSPs: a complexity map of (non-deterministic) multivalued decision diagrams, in: ICTAI, IEEE, 2012, pp. 1–8.
[3] Ola Angelsmark, Johan Thapper, A microstructure based approach to constraint satisfaction optimisation problems, in: Ingrid Russell, Zdravko Markov (Eds.), Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, Clearwater Beach, Florida, USA, AAAI Press, 2005, pp. 155–160.
[4] Richard Beigel, David Eppstein, 3-Coloring in time $O(1.3446^n)$: a no-MIS algorithm, in: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23–25 October 1995, IEEE Computer Society, 1995, pp. 444–452.
[5] Christian Bessière, Jean-Charles Régin, Roland H.C. Yap, Yuanlin Zhang, An optimal coarse-grained arc consistency algorithm, Artif. Intell. 165 (2) (2005) 165–185.
[6] Xinguang Chen, Peter van Beek, Conflict-directed backjumping revisited, J. Artif. Intell. Res. (JAIR) 14 (2001) 53–81.
[7] David A. Cohen, Martin C. Cooper, Páidí Creed, Dániel Marx, András Z. Salamon, The tractability of CSP classes defined by forbidden patterns, J. Artif. Intell. Res. (JAIR) 45 (2012) 47–78.
[8] Martin C. Cooper, Guillaume Escamocher, A dichotomy for 2-constraint forbidden CSP patterns, in: Jörg Hoffmann, Bart Selman (Eds.), AAAI, AAAI Press, 2012.
[9] Martin C. Cooper, Stanislav Živný, Tractable triangles and cross-free convexity in discrete optimisation, J. Artif. Intell. Res. (JAIR) 44 (2012) 455–490.
[10] Martin C. Cooper, Marie de Roquemaurel, Pierre Régnier, A weighted CSP approach to cost-optimal planning, Artif. Intell. Commun. 24 (1) (2001) 1–29.
[11] Martin C. Cooper, Peter G. Jeavons, András Z. Salamon, Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination, Artif. Intell. 174 (9–10) (2010) 570–584.
[12] Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, Bruno Zanuttini, On broken triangles, in: O'Sullivan [27], pp. 9–24.
[13] Martin C. Cooper, Efficient systematic analysis of occlusion, Pattern Recognit. Lett. 7 (1988) 259–264.
[14] Martin C. Cooper, Fundamental properties of neighbourhood substitution in constraint satisfaction problems, Artif. Intell. 90 (1–2) (1997) 1–24.
[15] Martin C. Cooper, Beyond consistency and substitutability, in: O'Sullivan [27], pp. 256–271.

[16] Gérard Cornuéjols, Xinming Liu, Kristina Vuskovic, A polynomial algorithm for recognizing perfect graphs, in: FOCS, IEEE Computer Society, 2003, pp. 20–27.
[17] Rina Dechter, Constraint Processing, Morgan Kaufmann Publishers, 2003, 340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205.
[18] Eugene C. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: Proceedings of AAAI-91, 1991, pp. 227–233.
[19] Jian Gao, Minghao Yin, Junping Zhou, Hybrid tractable classes of binary quantified constraint satisfaction problems, in: AAAI, 2011.
[20] Ian P. Gent, Christopher Jefferson, Ian Miguel, Watched literals for constraint propagation in minion, in: Frédéric Benhamou (Ed.), CP, in: Lect. Notes Comput. Sci., vol. 4204, Springer, 2006, pp. 182–197.
[21] M. Grötschel, L. Lovasz, A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, Combinatorica 1 (1981) 169–198.
[22] P. Jegou, Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems, in: Proceedings of the 11th National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, USA, Jul 1993, pp. 731–736.
[23] Javier Larrosa, Rina Dechter, Boosting search with variable elimination in constraint optimization and constraint satisfaction problems, Constraints 8 (3) (2003) 303–326.
[24] Christophe Lecoutre, Constraint Networks: Techniques and Algorithms, ISTE/Wiley, 2009.
[25] Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, Microstructures for CSPs with constraints of arbitrary arity, in: Alan M. Frisch, Peter Gregory (Eds.), Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation, SARA 2013, 11–12 July 2013, Leavenworth, Washington, USA, AAAI, 2013, pp. 11–12.
[26] Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, Bruno Zanuttini, Some new tractable classes of CSPs and their relations with backtracking algorithms, in: Carla P. Gomes, Meinolf Sellmann (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Proceedings of the 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18–22, 2013, in: Lect. Notes Comput. Sci., vol. 7874, Springer, 2013, pp. 61–76.
[27] Barry O'Sullivan (Ed.), Principles and Practice of Constraint Programming – Proceedings of the 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014, Lect. Notes Comput. Sci., vol. 8656, Springer, 2014.
[28] Patrick Prosser, Hybrid algorithms for the constraint satisfaction problem, Comput. Intell. 9 (3) (November 1993) 268–299.
[29] Francesca Rossi, Peter van Beek, Toby Walsh (Eds.), The Handbook of Constraint Programming, Elsevier, 2006.
[30] András Z. Salamon, Peter G. Jeavons, Perfect constraints are tractable, in: Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming, CP 2008, Sydney, Australia, 14–18 September, in: Lect. Notes Comput. Sci., vol. 5202, Springer, 2008, pp. 524–528.
[31] Rustem Takhanov, A dichotomy theorem for the general minimum cost homomorphism problem, in: Jean-Yves Marion, Thomas Schwentick (Eds.), STACS, in: LIPIcs, vol. 5, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010, pp. 657–668.
[32] Brian C. Williams, Robert J. Ragno, Conflict-directed A^* and its role in model-based embedded systems, Discrete Appl. Math. 155 (12) (2007) 1562–1595.